

VULNERABILITY ASSESSMENT

A Component-based Design using CORBA

Michael Cloppert

April, 2005

TABLE OF CONTENTS

<i>Abstract</i>	3
<i>Overview</i>	3
<i>Related Works</i>	4
<i>Background: Vulnerability Assessment</i>	4
<i>Overview</i>	4
<i>Current solutions</i>	5
<i>Unresolved Problems</i>	6
<i>Proposed Solutions</i>	7
<i>Background: CORBA</i>	8
<i>Overview</i>	8
<i>Simple Example</i>	9
<i>Application: Re-designing Vulnerability Assessment</i>	11
<i>Defining the Scan Task</i>	11
<i>CORBA Interface Definition</i>	12
<i>Building in Multithreading</i>	14
<i>The Final Product</i>	15
<i>Conclusion</i>	16

VULNERABILITY ASSESSMENT

A Component-based Design using CORBA

Michael Cloppert

April, 2005

Abstract

In this paper, methods of improving the design of network vulnerability assessment (VA) software with the Object Management Group's (OMG) CORBA component architecture will be investigated. Current COTS and open-source vulnerability assessment tools have no mechanism by which the rapid scanning of large pools of IP addresses can be distributed evenly and automatically across multiple systems. Distributed VA tools typically require that users manually assign target IP addresses to each physical scanning device performing assessments. Besides being prone to human error, this approach leads to scalability issues and limits the system's ability to efficiently handle network or system problems. The inherent accessibility, latency, and bandwidth inconsistencies of IP networks cause some scanning devices to finish their queues and become idle, while others have many addresses left to scan. If a scanning device fails, the static nature of this design prevents other scanning devices from assuming more tasks to ensure all IP addresses are scanned in an efficient manner.

By evenly distributing the vulnerability assessment tasks across multiple systems, these sorts of problems can be avoided. Two improvements are investigated: reducing the size of the vulnerability assessment tasks handed off to scanners, and dynamically assigning tasks between available scanners. The former is a simple change in the definition of a vulnerability assessment "task." The latter employs CORBA's component architecture for more flexible and adaptable task distribution.

Overview

After a brief discussion of related research, a background on vulnerability assessment tools will be presented. Current solutions, along with their shortcomings, will be analyzed. The analysis of the state of vulnerability assessment tools comes from my own personal experience, which includes implementing and using a variety of solutions in large environments over the past four years. Following this, CORBA will be discussed in brief, illustrating its strengths as a distributed component-based architecture and outlining its mechanics. The two approaches to improving upon enterprise VA software are then discussed: re-defining

the concept of the scan “task”, followed by an in-depth discussion of the appropriate CORBA interface definition.

Enterprise-class vulnerability assessment software is very complex. Analyzing the entire solution, from the ground up, is not within the scope of this paper. The reader should take away from this document an idea of how CORBA can be implemented in the design of a more effective enterprise-class vulnerability assessment product, not a finished vulnerability assessment solution.

RELATED WORKS

An abundance of research has been done on vulnerability assessment, particularly relating to improving the accuracy of vulnerability identification methods and remote system identification.¹ Likewise, significant research has been done on the use of CORBA in everything from real-time systems [2] to multi-threaded applications [3] and network performance [4]. The foundation has been laid for some of the key components of a CORBA-enabled, component-based vulnerability assessment solution; however, no public research on the topic was available as of the writing of this paper.

Background: Vulnerability Assessment

OVERVIEW

Vulnerability Assessment software is designed to assist network and system administrators in identifying networked hosts vulnerable to compromise. This makes identifying risk and prioritizing software updates much easier for administrators. Typically, enterprise-class vulnerability assessment falls into one of two categories: host-based and network-based systems. While host-based systems require software on each system that will be analyzed (referred to as *targets* in this paper), network-based vulnerability assessment software analyzes systems remotely. A target’s exposure is determined by its response to various network stimuli initiated by the system running VA scanning software, or *scanner*. Host-based systems can provide more information on a target and tend to be more accurate; however, these systems have higher deployment and maintenance costs. This paper will focus exclusively on network-based vulnerability assessment software.

Key challenges to network-based vulnerability assessment software can be broken down into the following categories: *scanning, storage, reliability, and dissemination*. The larger the environment in which a vulnerability assessment solution is deployed, the more difficult meeting

¹ While many private companies and public projects have done work in this area, much of the early, ground-breaking work was done by Fyodor in his nmap tool [1].

these challenges becomes. More networked hosts means a more heterogeneous pool of systems to scan and more time to scan all systems, more data to store in the database, and more individuals to whom data should be disseminated. With all of these increased complications comes a higher likelihood of problems.

CURRENT SOLUTIONS

Modern vulnerability assessment systems address challenges with a three-tier approach: a variable number of *scanners* that perform assessments, a *database* at the back-end, and an *application server* in the middle to coordinate scans, interface with the database, and provide an interface through which users can administer and control the system. More robust solutions allow the addition of any number of scanners, enabling parallel scanning of targets in larger environments. Figure 1 illustrates the relationship between these components.

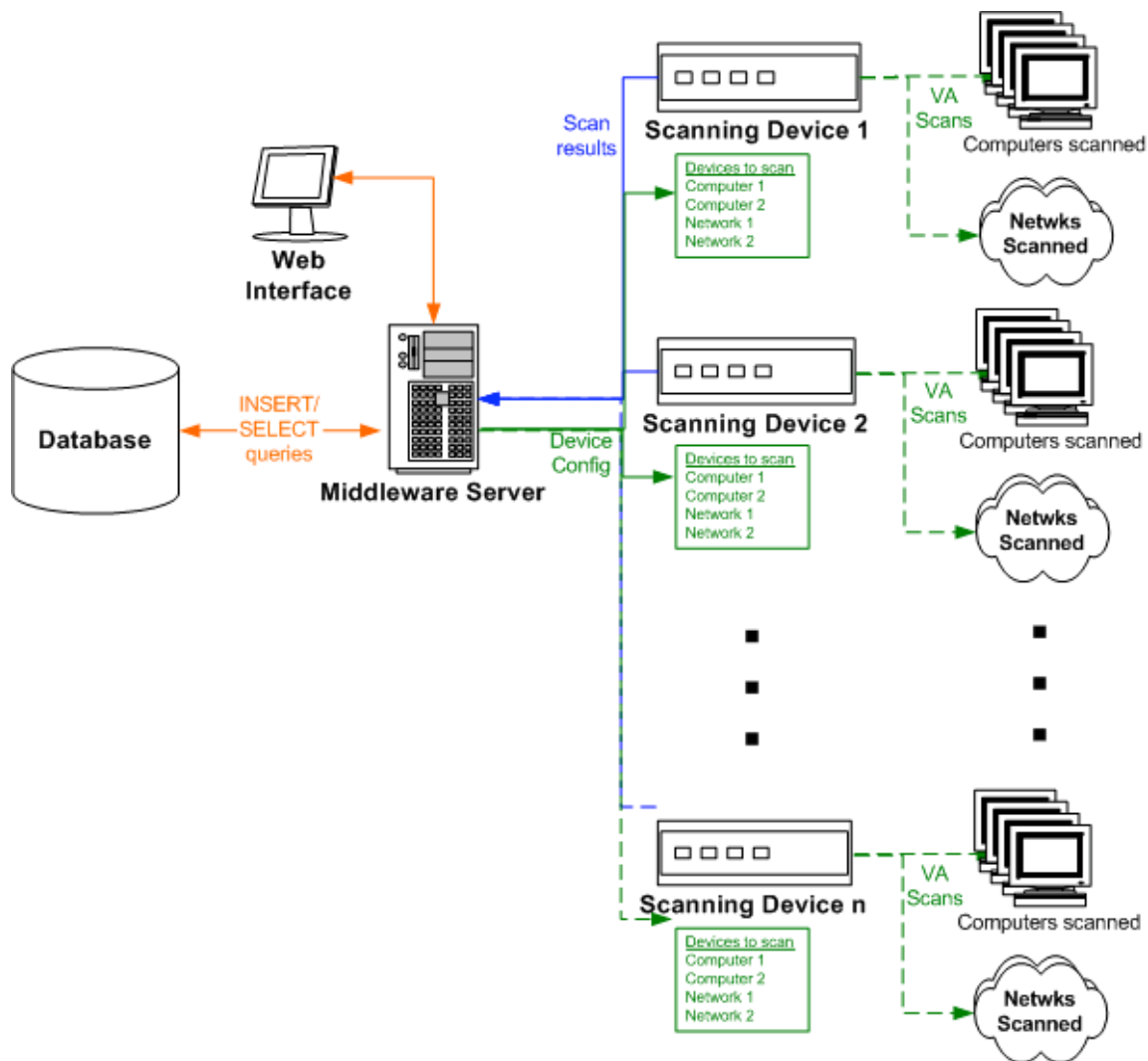


Fig. 1: The Three Tiers of Current Vulnerability Assessment Solutions

Database construction and maintenance are problems that have been researched and solved. For this reason, vulnerability assessment solutions should (and nearly always do) use a database provided by another vendor. This permits designers to focus on the application-specific problems presented by the design and maintenance of the scanners and middleware server.

UNRESOLVED PROBLEMS

Because databases are essentially a solved problem, the most significant outstanding issues in current vulnerability assessment solutions affect the two remaining tiers. Specifically, problems in the accuracy & efficiency of scan results and reliability of the overall system remain outstanding. The focus of this paper will be on the area in which proper implementation of component architecture such as CORBA can assist: improving the efficiency of scan results and reliability of the system.

Current industry-leading products divide the pool of hosts to be scanned amongst available scanners by IP address. That means the smallest atomic task handed off to a scanner includes all vulnerability scans available for a single IP address. These tasks are then assigned to scanners manually by administrators of the system. This approach leads to an inflexible system that cannot adapt when problems occur.

In large, distributed networks, anomalies like latency and packet loss are a very real problem. Also, older or very busy systems can respond slowly to queries by vulnerability assessment scanners. All of this entropy is magnified as the set of hosts to scan grows larger. Some scanners will take a longer time than expected to complete hosts in their queue as they are delayed waiting on slow hosts, while other scanners may experience less problems and complete their queues more quickly. This leads to a situation where some scanners are idle, while others have yet to complete their queue, thus reducing the overall efficiency of the system (See Figure 2). Reporting critical vulnerabilities to appropriate personnel is time-sensitive, so these delays need to be minimized.

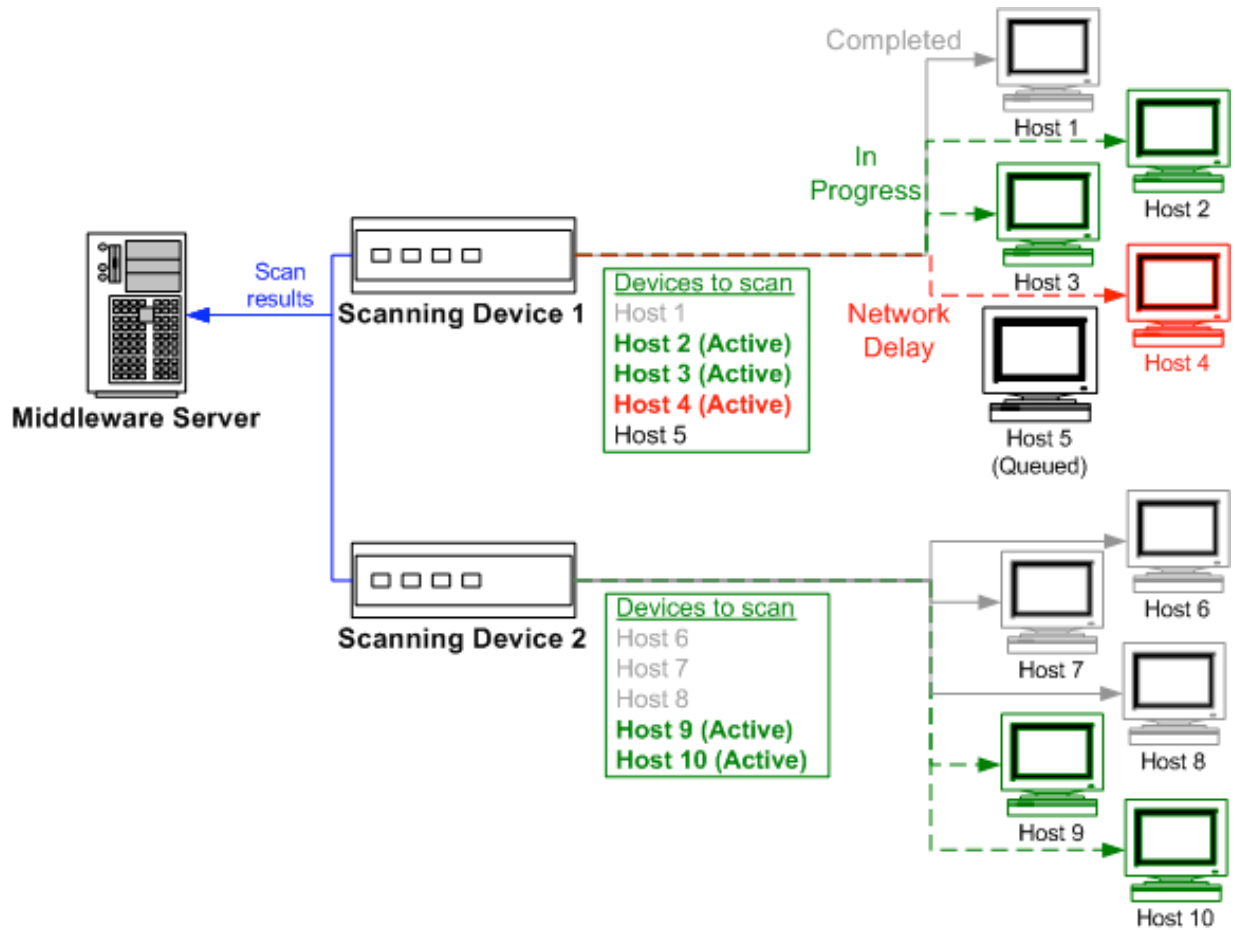


Fig. 2: Slow Network causes Delay in Queue Completion.

Larger problems relating to the system’s inability to move scan tasks between scanners can occur if one of the scanners experiences a problem or becomes unavailable. In this case, *none* of the scan tasks assigned to the problematic scanner will be executed, resulting in a gap in data. Furthermore, adding to or removing from the pool of available scanners requires significant maintenance, as IP’s need to be manually re-distributed.

PROPOSED SOLUTIONS

Quite obviously, re-designing the VA system to dynamically assign scan tasks to scanners based on availability would fix the problems introduced by latent scans and unavailable scanners. Decreasing the size of each scan task would enhance these improvements by increasing the speed and evenness with which the system could reassign tasks.

Background: CORBA

O V E R V I E W

CORBA is the Common Object Request Broker Architecture, a component architecture designed by the Object Management Group (OMG) as a specification for program interoperability between networked hosts. It was designed to be functionally similar to Microsoft's DCOM, Sun's Java Beans, etc., but free of vendor dependencies. There are two parts of the specification that are critical to this interoperability: the Interface Definition Language (IDL) and the underlying protocols GIOP and IIOP [4]. For two pieces of code to communicate using CORBA, they must use an identical IDL specifying the same input and output parameters in the proper order, as well as be able to use the common underlying protocols [5].

CORBA applications fall into one of two types: *clients* invoking operations on objects, and *servers* processing operations and returning the results. In order for a CORBA application to function, all objects on the server that provide operations for clients to invoke must have an IDL defined². The same IDL is used by the client to invoke operations on objects, as well as the server to receive requests and return the results of the invoked operations. However, the details of how the operation is executed are hidden from the client; all it sees is the interface. Figure 3 (labeled in the image as “Figure 1”) illustrates how these components work together. *IDL Stub* and *IDL Skeleton* are terminology used for the client and server sides of the IDL, respectively. The Object Request Broker, or ORB, handles not only the delivery of the request from client to server, but also uniquely identifies the request so that the server invokes an operation on the proper object, not an object that has been previously requested on by another client (or another thread on the same client).

² The IDL is not a programming language itself. The OMG provides mappings to many common languages, however [6].

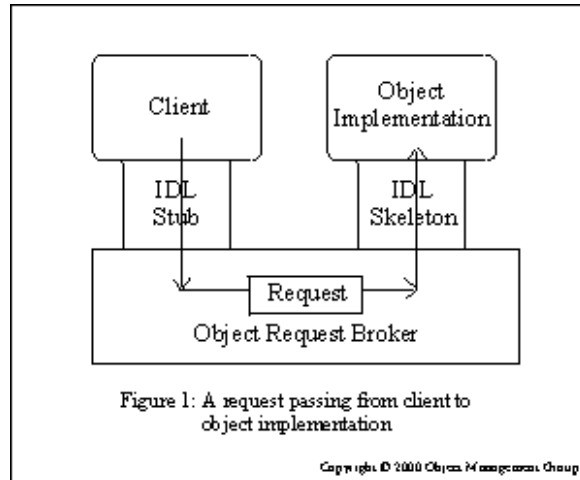


Fig. 3: Illustration of a Generic Object Request [5]

SIMPLE EXAMPLE

The process of creating an IDL and invoking operations on an object with CORBA is best described by example. The object that will be worked with will be a simple integer. The client will be allowed to set the integer value of the object (which resides on the server), and get the integer value of the object. This is not a very useful example, but it illustrates all of the key functions necessary to implement and use CORBA.³

- First, the interface is defined. The only functions necessary are to set and get the integer.

```
//someIntIDL.idl
interface someIntIDL {
    void set(in long i);
    long get();
};
```

- The object class must be defined as well. However, this will only be seen by, and known to, the server.

```
//someInt.h
class someInt {
public:
    void set(int i);
    int get();
private:
    int x;
};

//someInt.cpp
#include "someInt.h"
void someInt::set(int i){
    x = i;
}
```

³ This example borrows heavily from a document published by the University of Alabama [7]

```
int someInt::get(){
    return x;
}
```

- Now, the server is coded. The object must be imported, as well as the IDL.

```
//server.cpp
#include "someInt.h"
#include "iostream.h"
void main() {

    // Initialize the CORBA ORB for our object requests.
    CORBA::ORB_var orb = CORBA::ORB_init();
    // Initialize the Basic Object Adapter (BOA). This is a vanilla
    // Adapter design pattern that adapts from the servant programming
    // language (C++ here) to the Object Request Broker, or ORB. [8]
    CORBA::BOA_var boa = orb->BOA_init();

    // Create a new someInt object.
    someInt s;

    //Create the _tie version of someIntIDL to support delegation
    _tie_someIntIDL<someInt> tie_someInt(s, "someInt");

    // Export the newly created object.
    boa->obj_is_ready(&tie_someInt);
    cout << "someInt object is ready." << endl;

    // Announce server is ready to receive requests
    boa->impl_is_ready();
}
```

- Finally, the client is coded. The IDL is still necessary, but the someInt object is unknown to the client.

```
// client.cpp - client implementation
#include <iostream.h>
#include <stdlib.h>
void main() {
    int x;

    //Initialize
    CORBA::ORB_var orb = CORBA::ORB_init();
    CORBA::BOA_var boa = orb->BOA_init();
    someIntIDL_var s;
    s = someIntIDL::_bind("someInt");
    while (1)
    {
        cout << "Enter value to set: ";
        cin >> x;
        cin.ignore();
        s->set(x);
        cout << "The value you set was: ";
        cout << s->get() << endl;
    }
}
```

Application: Re-designing Vulnerability Assessment

DEFINING THE SCAN TASK

The smaller the scan task is, the more evenly it can be divided across scanners. Currently, the smallest atomic scan task involves performing all vulnerability scans available on a single IP address. Vulnerabilities of interest can number over 1,000. This means, in essence, each atomic scan task carries with it over 1,000 different operations or queries before the task is complete. This is a very large task to hand to the scanner. The possibility also exists that the vulnerabilities scanned on one IP address could greatly outnumber the vulnerabilities scanned on another. For instance, an old IBM AS/400 will have far fewer known vulnerabilities than a Windows 95 computer. Even assuming that scan tasks are automatically assigned to idle scanners, these two scans would not be evenly distributed. The scans are only broken down by IP address, meaning the scanner that receives the request to scan a Windows 95 computer will have a far larger task than the scanner that receives the request to scan the AS/400. The “critical path,” or shortest period of time that data from both scans would be available, is the period of time one scanner needs to perform all scans of the largest scan task. Furthermore, if a third scanner were available, and these were the only two hosts being scanned, it would remain idle.

By re-defining the atomic scan task to be the pair of {IP Address, Vulnerability Check}, the problems in the previous example would be eliminated. Tasks could be assigned to scanners as they become idle, and scanners would become idle more quickly, allowing for a more even distribution of scan tasks.

To compare the efficiency of the two approaches, assume T is the amount of time needed to complete any one scan. Also assume that there are three scanners available, 10 vulnerabilities to check on the AS/400 system, and 21 vulnerabilities on the Windows 95 system [9].⁴ The amount of time required for all tasks to be completed with the old atomic scan task is $21T$, even though the AS/400 is finished after $10T$. With the re-defined scan task, the scan tasks are broken up evenly across all three scanners, and all data is available after $(21T + 10T)/3 = 10.33T$. This is a performance improvement of 50.8%.⁵

⁴ Keep in mind these are only vulnerabilities in core OS components, not applications that run on them.

⁵ Of course, not all vulnerability scans are created equal, and some take significantly more time than others. Equal execution time was assumed to simplify the example.

CORBA INTERFACE DEFINITION

The heart of this document, of course, is employing CORBA in the design of vulnerability assessment software so as to improve efficiency, scalability, management, and error handling.

In CORBA terminology, the middleware server will act as the “client,” requesting scan objects from the scanners, which will act as “servers”

It has been shown that one of the key components is the CORBA interface definition, and properly using the definition in the IDL Stub and IDL Skeleton. The focus of this section will be on the interface definition and a simplified version of the IDL Stub and IDL Skeleton, building on the previous example. The details of the software on the scanner are intentionally omitted; it is assumed that this is a previously-solved problem. The scanIDL listed below takes into account the new atomic scan task, defining requests as a single vulnerability check against a single IP address.

```
//scanIDL.idl

// define O/S Types.  In reality, this would be a very, very long list.
enum os_t { Win95, WinNT, Win2k, WinXP, WinME, LINUX, AIX, AS400 };

// define IP protocol service types
enum ip_prot_t { tcp, udp };

// define scan result type
struct scanResult_t {
    // Is the host vulnerable?
    bool isVulnerable;
    // What was the host's response?  This helps in troubleshooting.
    char *hostResponse;
};

interface scanIDL {
    // Set atomic scan task of {IP, vulnerability scan}
    void set(in long ip, in short vuln_index_number);

    // Get O/S type
    os_t get_os();

    // Get running services, for either tcp or udp ports
    *short get_services();

    // Scan host for a single vulnerability
    scanResult_t scan_vuln();
};
```

Two interfaces that need to be provided to the server's scan object (not mentioned previously) are *get_os* and *get_services*. These two tasks are necessary for the client to determine which scans to run against a particular host, and are functionally quite different from probing a single vulnerability. As can be inferred from the IDL, each of these is treated as a single scan task. Also note that the list of operating system types in the IDL is abbreviated. An accurate listing of all OS types and subtypes would be quite lengthy, and specific to what the scanner has the capability to detect. The return value for *get_services* is an array of inte-

gers. Each integer will represent a TCP or UDP port that has been identified by the scanner as providing a listening “service” (TCP port 80 on a web server, for example). Note that the vulnerability status is returned by *scan_vuln*, along with the host’s response that caused the scanner to set or unset the *isVulnerable* boolean. Often, this information is needed by users to identify incorrectly-reported vulnerabilities, or other troubleshooting purposes. The *vuln_index_number* parameter assumes that the client and server both have identical, indexed lists of all vulnerability checks available.

The server will only be discussed insofar as its interface with the CORBA ORB. As a result, the server implementation has been significantly over-simplified. In the code below, it is assumed that the *scan_functions.h* class contains C++-equivalent definitions of the types *os_t*, *ip_prot_t*, and *scanResult_t* used in the IDL, as well as the functions *get_os*, *get_services*, and *scan_vuln*. It is also assumed that the scan object type is *scan_t*.

```
//server.cpp
// Scanner “server” providing scan functions

// include the functions already written to handle the logistics of scanning
#include scan_functions.h

int main() {
    // Initialize the CORBA ORB
    CORBA::ORB_var orb = CORBA::ORB_init();
    // Initialize the BOA
    CORBA::BOA_var boa = orb->BOA_init();

    scan_t scans;

    //Create the _tie version of scanIDL to support delegation
    _tie_scanIDL<scan_t> tie_scanIDL(scans, "scan_t");

    // Export the newly created object.
    boa->obj_is_ready(&tie_scanIDL);

    // Announce server is ready to receive requests
    boa->impl_is_ready();
}
```

All that happens in this piece of code is that the object *scan_t*, which contains all functions necessary for performing a *single* vulnerability scan on a *single* IP address, is made available to the ORB via the BOA. The client will also be based on the previous example.

```
// client.cpp
#include <iostream.h>
#include <stdlib.h>

void main() {

    struct scanResult_t {
        short isVuln;
        char *result;
    } result;

    int ip;
    int vuln_check;
```

```

//Initialize
CORBA::ORB_var orb = CORBA::ORB_init();
CORBA::BOA_var boa = orb->BOA_init();
scanIDL_var scanner;
scanner = scanIDL::_bind("scan_t");
while (1)
{
    cout << "Enter IP Address to scan";
    // Note that "IP" isn't in dotted decimal notation, but rather
    // in integer notation.
    cin >> ip;
    cout << "Enter vulnerability to scan for";
    cin >> vuln;
    scanner->set(ip, vuln);
    result=scanner->scan_vuln();
    if (result.isVuln) { cout << "This host is vulnerable!"; }
    else { cout << "This host is not vulnerable!"; }
}
}

```

This sample code shows a client serially requesting scans against for a specific vulnerability on a single IP addresses: the new atomic scan task. The user is prompted for an IP address, in integer notation, as well as an index for a vulnerability. Obviously, many assumptions are made here and this is not a very practical implementation.

BUILDING IN MULTITHREADING

In the most recent example using CORBA to open scan objects to clients, only one scan is requested at a time by the client, and only one scan object is made available at a time by the server. To take full advantage and justify the use of CORBA, the application should be designed to accommodate multiple threads at the client and server sides. The client should be able to request scan objects until all threads on all servers are consumed, maximizing scan efficiency. CORBA's mechanisms for multithreading will be discussed here, but not implemented, as multithreaded CORBA application design is a very complex topic in and of itself.

The Basic Object Adapter (BOA) used in examples thus far was designed as an adapter for C, and is not an effective adapter for C++. A more robust adapter, called the Portable Object Adapter, has been developed that includes the interfaces with C++ needed for multithreading (POA) [8, p.22]. This adapter should be used when designing more complex applications such as the vulnerability assessment application discussed here.

A POA on the target object - that is, the object on the server - is created with a particular threading policy. Enabling multithreading for the scanner involves creating the POA with the *ORB_CTRL_MODEL* threading policy. The requests for scan objects from the client will be relatively infrequent and long-running, making a thread-per-request model an ideal approach in implementing the POA. In this model, each incoming request to the server gets its own thread. When the maximum number of threads for the server has been reached, an exception can be thrown to let the client know that it should wait for some

other scans to complete before it requests another scan object [8, p. 975-976]. As more “servers” (or, in the case of our specific application, scanners) are added, the thread pool increases, and so does parallelism, decreasing the total runtime for all scans as shown earlier.

THE FINAL PRODUCT

With the scan task redesigned for automatic, dynamic delegation to a scanner by the middleware server, middleware and scanner software re-designed to use CORBA for requesting scan objects, and multithreading implemented so that scans run concurrently on scanners upon request from the middleware, the vulnerability assessment system effectively uses all of its available resources to process queues of scan tasks (Fig. 4)

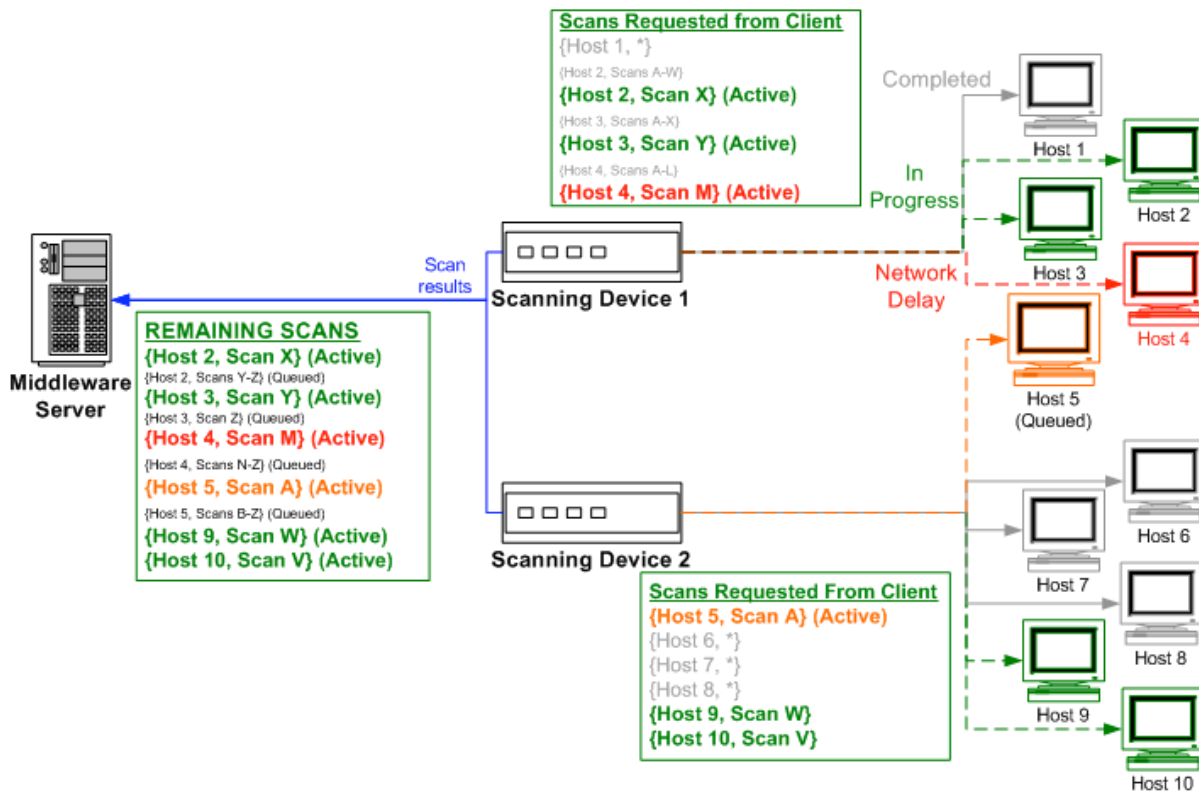


Fig. 4: How the re-designed system adapts automatically to a latency issue

In situations where scanners are unavailable, the client requesting scans isn't affected other than receiving thrown exceptions after fewer scan objects have been requested by the POA. The ORB handles the communication from client to the available servers, so when one goes offline, requests are simply not routed to it. Similarly, when scanners are added, the only difference seen by the client is the capability to request more scan objects before an exception is thrown. The solution is elegant and adaptable.

Conclusion

Current enterprise-class, network-based vulnerability assessment software contains a fundamental design flaw that limits its flexibility and can cause serious delays in obtaining time-critical information on vulnerable networked hosts. A carefully-specified CORBA interface definition and implementation, combined with a slight modification of the tasks handed to scanners, can address this shortcoming in its entirety by dynamically distributing scan tasks to available and functional scanners. Further research is necessary to validate such a change would not adversely impact other critical components of the system, and that all other requirements of vulnerability assessment software are met. Other changes would be required of the middleware, including rewriting procedures related to the assembly and storage of the scan results. Integrating this work with previous research on efficiently multithreading CORBA applications would yield a strong base from which effective solutions could be derived.

BIBLIOGRAPHY

- [1] Fyodor, *Nmap Network Mapper*,
<http://www.insecure.org/nmap>,
- [2] Krishna, Arvind S., Schmidt, Douglas C., Klefstad, Raymond, *Enhancing Real-time CORBA via Real-time Java Features*,
<http://www.cs.wustl.edu/~schmidt/PDF/RT-POA.pdf>.
- [3] Schmidt, Douglas C., *Implemented Multi-threaded CORBA Applications with ACE and TCO*,
<http://www.cs.wustl.edu/~schmidt/PDF/MT-CORBA4.pdf>.
- [4] Object Management Group, *Introduction to OMG Specifications*,
<http://www.omg.org/gettingstarted/specintro.htm>, 2005.
- [5] Object Management Group, *CORBA FAQ*,
<http://www.omg.org/gettingstarted/corbafaq.htm>, 2005.
- [6] Object Management Group, *CORBA Language Mapping Specifications*,
http://www.omg.org/technology/documents/formalcorba_language_mapping_specs.htm, 2005.
- [7] University of Alabama Department of Computer Science, *Integrating Component-Based Software Development into the Computing Curriculum*,
<http://cs.ua.edu/components/integrated/handouts/09-corb-a-example.pdf>.
- [8] Henning, Michi, and Vinoski, Steve, *Advanced CORBA Programming with C++*, Addison-Wesley, 1999.
- [9] Various, *The Open Source Vulnerability Database*,
<http://www.osvdb.org/>